VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Sběr a zobrazení procesních dat z výrobních PC

Data acquisiton and representation from industrial PC

2009 Petr Hanták

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 7. května 2009 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2009 .....

Abstrakt

Práce se zabývá tvorbou systému pro sběr a zobrazení dat z výrobních PC. Popisuje technologie využité při jeho tvorbě. Zaměřuje se na rozdíl mezi PUSH a PULL přístupem, porovnává typy datových souborů jako CSV, INI, XML. Rozebírá tvorbu XML dokumentu a jeho zpracování pomocí SAX parseru. Ukazuje a vysvětluje použití schémových jazyků DTD a XSD. Pomocí příkladu vysvětluje použití webových služeb v jazyce Java.

Klíčová slova: data z výrobních PC, XML, DTD, XSD, webové služby, PUSH a PULL přístup

Abstract

Bachelor thesis deal with data acquisiton and representation from industrial PC system. It describes technologies used in development this system. Compare PUSH and PULL access style and compare some data file types like CSV, INI, XML. This thesis analyze creating of XML documents and their parsing by SAX parser. It shows and explains using of scheme languages DTD and XSD. Via example explains using of web services in Java programming language.

Keywords: data from industrial PC, XML, DTD, XSD, web services, PUSH and PULL access style

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CSV	–	Comma Separated Values
DB	–	Database
DTD	–	Document Type Definition
EI	–	Equipment Integration
FTP	–	File Transfer Protocol
HDD	–	Hard Disk Drive
HTML	–	Hyper Text Markup Language
HTTP	–	Hypertext Transfer Protocol
IANA	–	(Internet Assigned Numbers Authority
IDE	–	Integrated Development Environment
IETF	–	The Internet Engineering Task Force
INI	–	Initialize file type
PC	–	Personal Computer
PDF	–	Portable Document Format
REST	–	Representational state transfer
RPC	–	Remote Procedure Call
SAX	–	Simple API for XML
SOAP	–	Simple Object Access Protocol
SŘBD	–	Systém řízení báze dat
W3C	–	World Wide Web Consortium
WS	–	Web Service
WSDL	–	Web Services Description Language
XML	–	eXtensible Markup Language
XPath	–	XML Path Language
XSD	–	XML Schema Definition

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V této práci se zabýváme systémem pro sběr a zobrazení dat z výrobních PC. Jednotlivé kapitoly popisují výběr, popis a použití vhodných technologií při jeho tvorbě.

Kapitola 2 rozebírá cíle práce a důvody, které vedou k tvorbě daného projektu. Taktéž ve zkratce popisuje, čím se firma ON Semiconductor zabývá a uvádí nás do problematiky spojené s výrobou polovodičů.

Uložením dat ze strojů do vhodných datových souborů se zabývá kapitola 3. V podkapitolách popisujeme jednotlivé datové typy a zejména v podkapitole 3.3 se zabýváme dokumenty typu XML. Protože právě tento typ byl použit v našem systému, je tato podkapitola nejdelší a snaží se typ XML přiblížit co nejvíce. Pro využití XML je potřeba v dnešní době spolupráce schémových jazyků. Tedy v dalších podkapitolách 3.3.1 a 3.3.2, rozebíráme schémové jazyky DTD a XSD.

V kapitole 4 rozebíráme možnosti vhodného přenosu souborů. Zaměřili jsme se na typ přístupu PUSH v podkapitole 4.1 a typ přístupu PULL v podkapitole 4.2. Oba tyto přístupy mají co nabídnout, ale volba toho správného pro daný problém nemusí být vždy snadná.

Kapitola číslo 5 porovnává způsob uložení dat. Podkapitola 5.1 popisuje výhody a nevýhody agendového zpracování dat, které se v dnešní době již prakticky nepoužívá (pokud ano, pak pouze na triviální případy). Agendové zpracování dat je uvedeno pouze pro přehled. Podkapitola 5.2 ukazuje výhody a nevýhody databázového zpracování dat. Protože jde o námi zvolený způsob, ukážeme si v další podkapitole 5.2.1 strukturu námi použité databáze. Popis obsahuje ER diagram spolu popisem jednotlivých tabulek.

O webových službách a jejich použití se dozvíme z kapitoly 6. V podkapitole 6.1 je popsán protokol SOAP a k čemu se ve webových službách používá. S tím je úzce spojena i podkapitola 6.2, která se zabývá jazykem pro popis webových služeb – WSDL. Pro pochopení je připojen i jednoduchý příklad implementace webové služby v programovacím jazyce Java.

Zpracováním XML dokumentů se zaobírá kapitola 7. Přestože zmiňuje i jiné způsoby, popisuje výhradně zpracování formou SAX parseru v programovacím jazyce Java.

Kapitola 8 popisuje technické řešení problému. Srozumitelnou formou se snaží ukázat spolupráci jednotlivých částí systému a rozepisuje se o jednotlivých aplikacích použitých jak na straně serveru, tak na straně klienta.

Dosažené poznatky a návrh na další rozšíření systému popisuje velmi stručně kapitola číslo 9.

2 Pozadí problému

Mezinárodní polovodičová firma ON Semiconductor sídlící v Phoenixu v Arizoně je jedním z předních světových výrobců integrovaných obvodů a diskrétních polovodičových součástek, které jsou používány v nejrůznějších elektronických zařízeních. Má své pobočky rozmístěny po celém světě. Část výrobních linek se nachází i v Rožnově pod Radhoštěm.

Polovodičová výroba je velice složitá. Než dostaneme finální výrobek, musíme materiál nechat zpracovat pomocí množství výrobních procesů. Výrobu je vhodné automatizovat a omezit tak co nejvíce chybu lidského faktoru. Stále to znamená potřebu operátora, který bude na výrobu dohlížet a reagovat na vzniklé události. Ve firmě se k ovládání strojů a řízení výroby využívá celá řada softwarových produktů.

Jedním z mnoha je i software zvaný Equipment Integration. Tento software na základě identifikace materiálu získává data z informačního systému. Tyto data zasílá pomocí komunikačního kanálu danému výrobnímu stroji a ten se dle nich nastaví.

Toto řešení zajišťuje pokročilou automatizaci celého výrobního procesu. Snaží se tak minimalizovat chybu způsobenou lidským faktorem. Kvalita výrobku se ověřuje pomocí přísného testování. I přes veškerou automatizaci a minimalizaci chyb způsobených člověkem, stále dochází k výrobě vadných kusů. Vadný kus nemusí byt odhalen při výrobě, nebo hned po jejím ukončení, ale například až o několik týdnů později. Firma se pomocí procesních inženýrů, kteří zajišťují kvalitu výrobního procesu na těchto strojích, snaží odhalit možnou chybu.

Momentálně není zpětné přezkoumání výrobního procesu určitého výrobku příliš transparentní. Údaje o samotném procesu sice jsou evidovány, avšak na samotných strojích. Dají se zpětně reprodukovat, ale je to zdlouhavé a nepřehledné. Equipment Integration umí se stroji komunikovat a měl by být schopen jednotlivé parametry (tlak, teplotu, rosný bod, napětí, proud, atd.) ze stroje pomocí komunikačního kanálu odeslat danému řídícímu PC.

Toho lze využít a všechna data ohledně celého průběhu zpracování daného materiálu někde přehledně evidovat. V praxi by to mělo vypadat například podle tohoto scénáře. Křemíková deska (vyrábí se řezáním z válcového monokrystalického ingotu) prochází postupně výrobou až dojde do difúzní pece. Ta bude mít předem nastaveno kolikrát se mají jednotlivé parametry při každém procesu sejmout. V průběhu procesu se postupně parametry ukládají do protokolu na HDD řídícího PC. Jakmile proces skončí, deska pokračuje k dalšímu zpracování a protokol se zašle do centrální evidence. K ní pak může procesní inženýr kdykoliv snadno přistoupit a dané parametry, získané v průběhu výroby, si zobrazit.

Tento scénář prozatím není zaveden a jeho návrh a uvedení do provozu je předmětem této práce.

3 Typy datových souborů

Stroje při výrobě odesílají data na výrobní PC. Abychom byli schopni data využít, musíme je uložit do datových souborů. Je potřeba zvolit typ datového souboru a navrhnout vhodnou strukturu uložení dat v tomto souboru. Nabízí se nám velice široké množství typů datových souborů. Samozřejmě lze vymyslet kompletně vlastní řešení. Různé, již používané typy mají své výhody. Mnohé z nich jsou například standardizovány. Dá se předpokládat, že některé typy programů si tak s nimi mohou lehce poradit, taktéž se v nich může běžný uživatel lépe orientovat. Dále mají například podporu různých API pro práci s nimi. To velice ulehčí programové zpracování a nemusí se vymýšlet znova již dávno naprogramované konstrukce. Srovnáme zde výhody a nevýhody tří nejpoužívanějších typů a navíc rozebereme klady a zápory námi vytvořeného vlastního typu.

3.1 Soubory typu CSV

Zkratka CSV znamená Comma separated values. Tedy česky hodnoty oddělené čárkou. Jde o jednoduchý textový souborový formát sloužící k výměně tabulkových dat. Soubory tohoto formátu můžeme nejčastěji nalézt pod různými příponami (nejčastěji *.csv* nebo *.txt*). Tato technologie je stará už od roku 1967, kdy byl podporován v kompilátoru IBM Fortran. Pro tento formát nenajdeme žádnou specifikaci, která by ho přesně definovala. Díky tomu se v mnoha aplikacích lehce odlišuje a sladění mezi dvěmi aplikacemi může být dosti obtížné. Pro CSV používané v internetové komunikaci můžeme nalézt podrobný popis formátu v doporučení RFC 4180[1] vydaném IETF (The Internet Engineering Task Force). Ten popisuje text/csv MIME type registrovaný u organizace IANA (Internet Assigned Numbers Authority).

Způsob zpracování tabulkových dat v tomto souborovém formátu si nejlépe ukážeme na jednoduchém příkladu.

Hodnoty jak je vidíme v tabulce (1), můžeme zapsat v CSV souboru pomocí tohoto zápisu níže. Všimněme si, že víceslovné řetězcové hodnoty jsou psány v uvozovkách. Chceme-li zapsat víceslovný řetězec obsahující uvozovky, vložíme je tak, že je zapíšeme dvakrát za sebou. Pokud máme neceločíselné hodnoty, píšeme tečku místo desetinné čárky, protože čárka je používána k oddělení hodnot. Některé programy, jako například Microsoft Excel, pracují i s národním nastavením. Používají normálně desetinnou čárku a na oddělení hodnot využívají například středník. Pokud se chceme vyhnout nekompatibilitě, je nejvhodnější používat čárku na oddělení hodnot a na neceločíselné hodnoty desetinnou tečku.

```
Leden,Ostrava,E350,450.52
Leden,"Washington ""USA""",1550.00
Únor,Ostrava,379.00
```

Výpis 1: Ukázka použití CSV

[1] RFC 4180 bylo vydáno v říjnu 2005 a jeho popis je možno nalézt na internetové adrese http://tools.ietf.org/html/rfc4180

Měsíc	Pobočka	Průměrně kusů
Leden	Ostrava	450.52
Leden	Washington "USA"	1550.00
Únor	Ostrava	379.00
...	...	...

Tabulka 1: Jednoduchá tabulka

Jak je možné vidět, samotný zápis v souboru je značně nepřehledný. Data jsou na řádcích co nejvíce zhuštěna. Bez předlohy, co znamená který sloupec, jsou pro člověka údaje nesmyslné. I když máme klíč, jsou údaje v souboru tak nepřehledné, že jejich čtení bez převedení do lepší formy je příliš zdlouhavé. Odhalit tak chybu v souboru obsahujícím stovky až tisíce záznamů, je velice obtížné. Pro programové zpracování může být obtížnější. U některých programovacích jazyků je potřeba napsat si kompletní obsluhu zpracování tohoto typu souborů. V případě Javy můžeme nalézt i jednoduchý Parser na tento typ souborů, který není však obsažen v základní distribuci (zde je třeba už řešit problém licencí). Dále zde nastává problém validace souboru. Jelikož není soubor generován podle nějaké šablony, nedá se ani dle žádné kontrolovat. To nám opět přidá více práce při implementaci a pokud lehce pozměníme strukturu dat, bude se muset upravit i samotný zdrojový kód na všech místech, kde se s CSV souborem pracuje.

Velkým kladem pro použití tohoto typu, je jeho rozšířená podpora zejména v tabulkových procesorech a importech/exportech dat do/z databází. Mezi nevýhody patří nepřehlednost a nesrozumitelnost pro běžného uživatele a malá flexibilita při změně struktury souborů.

3.2 Soubory typu INI

Formát INI souborů se začal masově využívat příchodem Microsoft Windows 3.x. To, co nám dnes v nových verzích Windows zajišťují registry, ve Windows 3.x obstarávaly právě INI soubory. Jak už název INI napovídá, jedná se o textové inicializační soubory. Soubory slouží k nastavení různých hodnot v daných programech. Jednoduchou změnou dané hodnoty právě v tomto souboru můžeme konfigurovat nastavení. Je to rychlé a jednoduché. Nic se nemusí složitě prohledávat. Toto sympatické řešení se tedy lehce uchytilo a díky tomu je dodneška podporováno. Spousta programů a aplikací na platformě Windows má právě ve svých adresářích tyto konfigurační soubory. Můžeme je nalézt pod různými koncovkami (*.ini*, *.cfg*, *.conf*, *.txt* a další).

Soubor formátu INI musí splňovat několik důležitých náležitostí, aby mohla být zaručena kompatibilita. Jednotlivé parametry se zapisují formou klíče a jeho hodnoty.

```
jméno = hodnota
```

Parametry také mohou být seskupeny do skupin pomocí takzvaných sekcí. Příkaz sekce je ohraničen hranatými závorkami a parametry, které za tímto příkazem bez-

prostředně následují, do této se sekce patří. Sekci lze ukončit dvěma způsoby. Novým příkazem sekce, nebo koncem souboru. Neexistuje žádná značka, která by sekci ukončila.

```
[název_sekce]
```

Samozřejmě můžeme psát i komentáře. Ty začínají středníkem a text, který následuje až do konce řádku, je komentářem. Provedení se středníkem je nejpoužívanější, lze však narazit i na komentář začínající znakem „#“. Je vhodné stanovit si předem, jak budou komentáře definovány.

V INI formátu se také vyskytují speciální znaky (escape sequence). Některé znaky, jako například zpětné lomítko, mohou zapříčinit chybu při čtení souboru. Pokud bude zpětné lomítko na konci řádku, může se stát, že bude konec řádku ignorován. Chceme-li tedy používat zpětné lomítko, zapíšeme jej jako dvě zpětná lomítka za sebou. Podobně je tomu i u ostatních escape sekvencí. Vždy začínají zpětným lomítkem a znakem/znaky pro danou sekvenci.

Problém prázdných řádků a nadbytečných mezer je komplikovanější. Zde už mnohdy závisí na samotné implementaci. Nejlepší způsob jak zaručit kompatibilitu je nedělat zbytečné mezery a netvořit prázdné řádky. S mezerami si programy poradí většinou tak, že je ignorují a víceslovné hodnoty musí být vloženy v uvozovkách. Ukažme si jednoduchý příklad souboru.

```
; nemanipulujte se souborem dokud je aplikace spustena
[general]
name = user2
plevel = 2
language = czech
[options]
mouse = yes
invert = no
resolution = 1024x768
```

Výpis 2: Ukázka použití zápisu v INI souboru

Tento formát opět neobsahuje možnost vytvořit si předpis pro validaci. Oproti CSV je ale mnohem přehlednější a snadno čitelný jak pro stroj, tak pro lidi. Lépe se v něm dá nalézt chyba. Velkou výhodou je podpora v mnoha programovacích jazycích. Existuje parser Nini[2] pro C#, pro jazyk C/C++ je to SDL Config[3], pro Javu se jedná o knihovnu ini4j[4].Tyto nám mohou zajistit pohodlnou práci s se soubory formátu INI.

Pomocí tohoto formátu můžeme snadno konfigurovat aplikace a ukládat menší množství dat. Pro uložení většího objemu dat, stojí za zvážení jiné pohodlnější a flexibilnější formáty.

[2] Více k Nini projektu na http://sourceforge.net/projects/nini

[3] SDL Config projekt – http://student.agh.edu.pl/~koshmaar/SDL_Config

[4] Java API pro obsluhu Windows INI file formátu – http://ini4j.sourceforge.net

3.3 Soubory typu XML

XML (eXtensible Markup Language) je značkovací jazyk. Značkovací jazyky jako takové, se využívají již po několik desítek let. „Asi prvním známým značkovacím jazykem byl GML (Generalized Markup Language), který vytvořili Charles Goldfarb, Edward Mosher a Raymond Lorie při práci na systému pro uchovávání a následné využití právních textů pro IBM.“[1, strana 13]

Po spojení GML se standardním formátovacím jazykem GenCode, vznikl roku 1986 jazyk SGML (Standard Generalized Markup Language). Tento jazyk je velice obecný a má široký záběr v mnoha oblastech. Jazyk pro tvorbu internetových stránek – HTML (Hyper Text Markup Language) je právě aplikací jazyka SGML. Jazyk HTML je omezen přesným počtem značek. Chybí možnost definovat si vlastní značky. A toho se dosáhlo právě vytvořením jazyka XML a XHTML (mutace právě pro tvorbu webu). Tyto jazyky byly stvořeny vybráním nejdůležitějších a nejpoužívanějších částí jazyka SGML. Je důležité mít na paměti, že jazyku XML jde o význam částí textu, ne o vzhled, jak je tomu například u jazyka HTML.

Co si můžeme pod tím vším představit? Jazyk XML nám umožňuje členit text podle našich potřeb. Můžeme zpracovávat jakýkoliv text. At’už od jednoduchého katalogu CD, nákupního lístku, až po různé tabulkové hodnoty. Vše bude snadno čitelné a srozumitelné jak pro stroj, tak pro člověka díky hierarchické stromové struktuře XML dokumentu. Velikou výhodou je otevřenost tohoto formátu. Nezávisí na platformě a ani na lokálních úpravách nástrojů. Specifikace XML je všude stejná [3].

I jednoduchou tabulku jako je tabulka (2), si můžeme zobrazit velice snadno jako XML dokument.

Os. číslo	Příjmení	Jméno	Ročník
M1021	Tišný	Jan	1
M1022	Malásek	Petr	3
M1023	Šťastná	Jarmila	1

Tabulka 2: Jednoduchá tabulka studentů

```
<?xml version="1.0" encoding="utf-8"?>
<skola>
    <student>
        <id>M1021</id>
        <prijmeni>Tišný</prijmeni>
        <jmeno>Jan</jmeno>
        <rocnik>1</rocnik>
    </student>
    <student>
        <id>M1022</id>
        <prijmeni>Malásek</prijmeni>
        <jmeno>Petr</jmeno>
        <rocnik>3</rocnik>
    </student>
```

```
    <student>
        <id>M1023</id>
        <prijmeni>Šťastná</prijmeni>
        <jmeno>Jarmila</jmeno>
        <rocnik>1</rocnik>
    </student>
</skola>
```

Výpis 3: Tabulka studentů v jazyce XML

Z výpisu (3) můžeme vidět, že každý XML dokument začíná nutnou hlavičkou, určující kódování dokumentu a samotnou verzi jazyka XML. Hlavička může obsahovat i další náležitosti, jako například šablonu značek. Každý dokument dále musí obsahovat právě jeden kořenový element. V našem případě je to element <skola>. Elementy vnořené v tomto elementu mohou mít již libovolný počet duplicit. XML jazyk rozlišuje malá a velká písmena, tedy názvy elementů <Student> a <student> nejsou totožné a budou brány jako dva různé. V názvech elementů se mohou vyskytovat i neobvyklé znaky, jako například pomlčky a tečky (pokud ale nechceme mít potíže při zpracovaní dokumentu, je lepší se těmto znakům vyhnout).

Značky - tagy, dělíme na párové a nepárové. Párové tagy mohou obsahovat další vnořené elementy na rozdíl od tagů nepárových, které mohou samy o sobě obsahovat pouze množinu atributů. V našem příkladu se vyskytují tagy pouze párové (mají začínající a ukončovací tag). Nepárový tag by vypadal takto: <kos hrusky="10"jablka="10"/>. Tedy element `kos` má dva atributy – `hrusky` a `jablka`, obojí o hodnotě deset. Atributy jsou využívány k upřesnění elementu. Tedy ne každý koš musí obsahovat stejný počet jablek a hrušek. Atributy lze použít i u párových tagů. Tam jsou definovány, vždy v počátečním tagu.

Jelikož samotné XML nás nechá využít tagů naprosto dle naší vůle, je vhodné celé spektrum nějak omezit. Vytvořit si nějakou šablonu pro používané tagy. Ta nám pomůže s validací samotného dokumentu a usnadní tak spoustu práce.

3.3.1 DTD

Definice typu dokumentu (DTD). Jedná se o schémový jazyk. Co znamená pojem schémový jazyk? Díky schémovému jazyku můžeme nadefinovat nový značkovací jazyk. Zjednodušeně omezíme kompletní množinu značek XML dokumentu na námi zvolené, které budou přesně odpovídat naší definované specifikaci.

Historie využití DTD v XML se datuje téměř k počátku působení XML. I když je DTD v dnešní době považováno za více než zastaralé, stále je masivně podporováno a využíváno. V dnešní době je však nahrazeno již pokročilejšími schémovými jazyky, jako je XML Schema. DTD samo o sobě nemá příliš možností oproti těmto mladším nástupcům.

Velice závažným nedostatkem DTD je žádná podpora jmenných prostorů. Při rozsáhlých XML dokumentech tak můžeme mít spoustu problémů. Mezi další velice důležité nedostatky patří nemožnost popisovat datové typy. Nelze tak rozlišit základní datové typy používané v objektovém zpracování a mnohdy nám to tak může zbytečně ztížit práci, kterou bychom ušetřili díky ošetření datových typů již v XML dokumentu.

3.3.1.1 Provázání DTD a XML dokumentu

Existují dvě možnosti, jak provázání dosáhnout. Lze například definovat DTD přímo v daném XML dokumentu. Toho se většinou nevyužívá, neboť při využití stejného DTD ve větším množství dokumentů, jde o plýtvání prostorem. Mnohem šikovnější je umístění DTD v externím souboru (nejčastěji s příponou *.dtd*). Díky tomuto řešení, můžeme stejné DTD využít v několika dokumentech naráz a při úpravě DTD, stačí pozměnit specifikaci pouze v daném externím souboru.

Provázání pomocí externího souboru se provádí takto:

```
<!DOCTYPE koren SYSTEM "jmeno_souboru.dtd">
```

Pomocí tohoto tagu, následujícího bezprostředně po hlavičce dokumentu, tedy načteme DTD z externího souboru. Všimněme si také slova "koren"v tomto zápisu. Na tomto místě musí být vždy vložen název kořenového tagu daného XML dokumentu.

Druhou možností je vložení DTD specifikace přímo do daného XML dokumentu, to se provádí následujícím způsobem:

```
<!DOCTYPE koren [
....zde jsou DTD specifikace...
]>
```

Existuje zde i další zajímavá možnost a to kombinace obou předchozích. Tedy specifikace pomocí externího DTD souboru s rozšířením této specifikace přímo v daném XML dokumentu. Toho docílíme tímto způsobem:

```
<!DOCTYPE koren SYSTEM "jméno_souboru.dtd" [
....zde jsou rozšiřující DTD specifikace...
]>
```

3.3.1.2 Deklarace

Deklarovat můžeme hned čtyři různé typy:

- elementy
- atributy
- entity
- notace

My si zde lehce přiblížíme deklaraci elementů a atributů, protože jde o nejčastěji používané prvky. O zbylých se můžeme snadno dozvědět více na internetových stránkách organizace W3C, kde lze nalézt i šikovné tutoriály pro uvedení do dané problematiky.

Deklarace elementu má jednoduchý tvar :

```
<!ELEMENT název obsah>
```

Název elementu musí začínat písmenem, ale následující znaky mohou být libovolné, záleží však na velikosti písmen. Obsah nám říká, co element může obsahovat. Nejlépe nám k vysvětlení poslouží malý příklad (4). Všimněme si notace jako u regulárních výrazů. Symboly "`?, *, +`" vyjadřují počet výskytů. Otazník znamená, že je výskyt nepovinný, znak plus znamená výskyt alespoň jednou a hvězdička znamená libovolný počet opakování. Bez použití těchto znaků definujeme výskyt právě jednou.

Veškeré další deklarace by měly být srozumitelné z komentářů v dané ukázce. Je třeba poznamenat , že definice `ANY` se příliš často nevyužívá, protože je trošku v rozporu s účelem vylisování. Definice je příliš benevolentní a přidá hodně práce při zpracování dokumentu. Doporučuje se jejímu použití spíše vyhnout.

```
<!-- element "jedna" nesmí nic obsahovat -->
<!ELEMENT jedna EMPTY>

<!-- element "dve" může obsahovat jak text, tak jiné elementy -->
<!ELEMENT dve ANY>

<!-- element "tvary" obasahuje jeden element "kruh" a jeden element "ctverec" -->
<!ELEMENT tvary (kruh, ctverec)>

<!-- element "tvar" obsahuje buď element "kruh", nebo element "ctverec" -->
<!ELEMENT tvar (kruh | ctverec)>

<!-- element "kniha" obsahuje element "nazev", libovolný počet elementů "autor", alespoň jeden
     element "kapitola" a může obsahovat element "stran"-->
<!ELEMENT kniha (nazev, autor*, kapitola+, stran?)>

<!-- element "text" obsahuje pouze text -->
<!ELEMENT text (#PCDATA)>
```

Výpis 4: DTD definice elementů

Pouze nadefinované elementy by nám stačily u jednoduchého dokumentu. Většinou chceme samotné elementy upřesnit více. K tomu nám slouží Atributy. Atributy a jejich obsah můžeme v DTD definovat podobně jako elementy. Všechny atributy pro daný element se definují za pomoci jednoho příkazu `ATTLIST` (list atributů). Jeho předpis je:

```
<!ATTLIST název_elementu deklarace_atributů>
```

Deklarace atributu se skládá z jeho názvu, typu, standardní hodnoty a případně jeho povinnosti. U názvu platí to samé, co u názvů elementů. Typ atributu probereme později. Definování standardní hodnoty a zvláště povinnosti atributu je důležité a může velice pomoci při validaci. Ušetří programátorovi při implementaci hodně práce. Obecně všechny možné chyby, které lze odhalit již validací, ušetří spoustu práce.

Kompletní předpis i s jednoduchým příkladem může vypadat takto:

```
<!ATTLIST název_elementu název_atributu typ_atributu default_hodnota>

<!-- Příklad deklarace atributu -->
<!ATTLIST literaura typ CDATA "naucna">
```

```
<!-- Ukázka v XML -->
<literatura typ="naucna" />
```

Mohli jsme si všimnou typu `CDATA`. Jde asi o nejobecnější typ atributu. Díky němu můžeme být hodnotou atributu libovolný textový řetězec. Samozřejmě máme ale k dispozici celou řadu typů atributu, jak můžeme vidět v tabulce (3).

Typ	**Popis typu atributu**
CDATA	Textový řetězec.
(en1—en2—...)	Enumerace - hodnota musí být z dané nabídky.
ID	Hodnota unikátního id v celém dokumentu.
IDREF	Hodnota id jiného elementu v dokumentu.
IDREFS	Seznam id jiných elementů v dokumentu.
NMTOKEN	Hodnotou je jedno slovo (stejné omezení jako ve jménech elementů a atributů).
NMTOKENS	Více slov oddělených mezerou, vyhovujících definici NMTOKEN.
ENTITY	Jméno entity.
ENTITIES	Seznam jmen entit.
NOTATION	Hodnotou je jméno notace.
xml:	Hodnota je předefinovaná xml hodnota.

Tabulka 3: Přehled možných typů atributů v DTD

Významy `CDATA`, `NMTOKEN` popřípadě `NMTOKENS` jsou si hodně podobné a z tabulky jasně čitelné. Chceme-li například u atributu „váha“ definovat hodnotu v kilogramech, je nejvhodnější využít `NMTOKEN` a hodnotu „10kg“ snadno nadefinujeme. Vždy bude očekáváno pouze jedno slovo.

Typy `ID`, `IDREF`a `IDREFS` se používají k odkazům v rámci dokumentu. Toho lze velice šikovně využít s použitím například jazyka XPath. Díky němu můžeme jednoduše provádět dotazy nad dokumentem. Zde se nebudeme touto problematikou zaobírat. Specifikaci jazyka XPath nalezneme na této internetové adrese[5].

`ENTITY` a `ENTITIES` typy slouží k ulehčení práce. Používají se v případě, kdy atribut obsahuje jako jméno hodnotu entity.

Entity jsou proměnné obsahující text nebo speciální znaky. Veliké využití mají při vkládání stejného úseku textu na více místech v dokumentu. Jsou dva typy entit - interní a externí. Interní můžeme definovat přímo a externí načítáme z jiného souboru. Jak se to provádí si ukážeme na jednoduchém příkladu:

```
<!--Předpis interní entity. -->
<!ENTITY jméno_entity "hodnota_entity">

<!--Předpis externí entity. -->
<!ENTITY jméno_entity SYSTEM "URI/URL">
```

[5]Specifikace jazyka XPath - http://www.w3.org/TR/xpath20/

```
<!−− Ukázka obou typů definic. −−>
<!ENTITY nazev "Dobrá kniha.">
<!ENTITY autor SYSTEM "http://nejaka_adresa/autori.dtd">

<!−− Ukázka v XML, všimněme si volání entit. −−>
<kniha>&nazev;&autor;</kniha>
```

Ještě jsme zde nerozvedli poslední důležitou definici atributu a to standardní hodnotu (default-value). Ta může nabývat hodnot podle tabulky (4).

Hodnota	**Popis**
`value`	Defaultní hodnota atributu.
#REQUIRED	Atribut je povinný.
#IMPLIED	Atribut není povinný.
#FIXED `value`	Atribut má pouze fixní definovanou hodnotu.

Tabulka 4: Přehled možných dafaultních hodnot atributů v DTD

Jak vidíme, nejedná se o žádné složité definice. Buď se k atributu vloží defaultní hodnota, ale ta nemusí být použita, nebo se řekne že atribut je povinný (musíme zadat hodnotu), či nepovinný (nemusíme zadat hodnotu). Poslední případ s fixní hodnotou by se dal vysvětlit jako použití konstanty. Atribut nemůže mít jinou hodnotu než standardní.

Ve výpise (5) můžeme vidět komplexní provázání definic elementů spolu s definicemi atributů. Je uveden i krátký příklad validního XML dokumentu pro takto definované DTD.

```
<!−− Definice elementů a jejich atributů. −−>
<!ELEMENT seznam ( firma* )>
<!ELEMENT firma ( zamestnanec* )>
<!ELEMENT zamestnanec EMPTY>
<!ATTLIST zamestnanec
          id  NMTOKEN #REQUIRED
          jmeno CDATA #REQUIRED
          oddeleni (IT|UCETNI) "IT">
<!ATTLIST firma
          ico  NMTOKEN #REQUIRED
          zamestnancu CDATA #IMPLIED>

<!−− Příklad validního XML dokumentu. −−>
<seznam>
  <firma ico="111" zamestnancu="1">
    <zamestnanec id="1" jmeno="Petr Rybka" oddeleni="IT" />
  </firma>
  <firma ico="112">
    <zamestnanec id="1" jmeno="Pavel Smutný" oddeleni="IT" />
    <zamestnanec id="2" jmeno="Jan Kopřiva" oddeleni="UCETNI" />
  </firma>
</seznam>
```

Výpis 5: DTD ukázka komplexní definice elemntů a atributů

3.3.2 XSD

Oproti DTD je schémový jazyk W3C XML Schema nebo též XSD (XML Schema Definition) nejvíce využívaným schémovým jazykem. Odstraňuje nevýhody DTD a dodržuje na rozdíl od DTD konvenci XML. To znamená, že ho lze například validovat proti sobě samému. Dále plně podporuje jmenné prostory a specifikace atributů a elementů může být opravdu dosti přísná. To je samozřejmě další plus.

XSD se používá od roku 2000 a momentálně ve verzi 1.0. Stále se aktivně pracuje na verzi 1.1 a více informací o této chystané verzi můžeme nalézt na internetových stránkách[6].

3.3.2.1 Provázání XSD a XML dokumentu

Oproti DTD je provázání XSD souboru (většinou přípona *.xsd*) a XML dokumentu složitější. Definice se provádí v kořenovém elementu XML dokumentu. Máme-li XSD soubor bez jmenného prostoru, je přehlednost této definice ještě slušná. Definujeme-li zároveň jmenný prostor, nebo dokonce více jmenných prostorů, pak je definice provázání velice nepřehledná. Avšak je to pouze malá cena za výhody, které nám to přináší. Můžeme tak nadefinovat hned několik XSD souborů najednou. Jak provázání vypadá, vidíme ve výpise (6). Druhý příklad byl převzat z tutoriálu W3C Schools[7]. Díky tomu je možné v ukázce vidět definici právě jejich jmenného prostoru. Může však být i jiný neevidovaný oficiálně [4].

```
<!-- Provázání bez jmenného prostoru. -->
<nazev_korenoveho_elementu
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="nazev_souboru.xsd">

<!-- Provázání spolu se jmenným prostorem. -->
<nazev_korenoveho_elementu
    xmlns="http://www.w3schools.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3schools.com nazev_souboru.xsd">
```

Výpis 6: Provázání XSD a XML dokumentu

3.3.2.2 Deklarace

Deklarovat XSD se dá několika způsoby. Popíšeme si nejjednodušší a nejpřehlednější (pozor rozhodně ne nejúspornější) metodu. Deklarovat budeme od těch nejjednodušších součástí XML dokumentu a propracujeme se až na nejvyšší hladinu [2].

Nejdříve se definují atributy a neprázdné koncové elementy. Vytvoříme tak nové, jednoduché datové typy. Ty budeme postupně skládat do složených datových typů tak dlouho, až se dostaneme ke kořenovému elementu XML dokumentu. Není to tedy nic složitého.

[6] Vývoj XSD verze 1.1 - http://www.w3.org/XML/Schema#dev
[7] XML Schema Tutorial - http://www.w3schools.com/Schema/schema_howto.asp

XSD má jednoduchý datový typ `xs:simpleType` a složený datový typ `xs:complexType`. Samozřejmě složený datový typ se skládá právě z typů jednoduchých. Ze všeho nejdříve se podíváme na definici elementů ve výpise (7).

```
<?xml version="1.0"?>

<!-- Povinná definice. -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Definice firmy. -->
<xs:element name="firma">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nazev" type="xs:string"/>
      <xs:element name="adresa" type="xs:string"/>
      <xs:element name="telefon" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>

<!-- Výsledný XML dokument. -->
<firma>
   <nazev>JmenoFirmy</nazev>
   <adresa>Ostrava...</adresa>
   <telefon>111-111-111</telefon>
</firma>
```

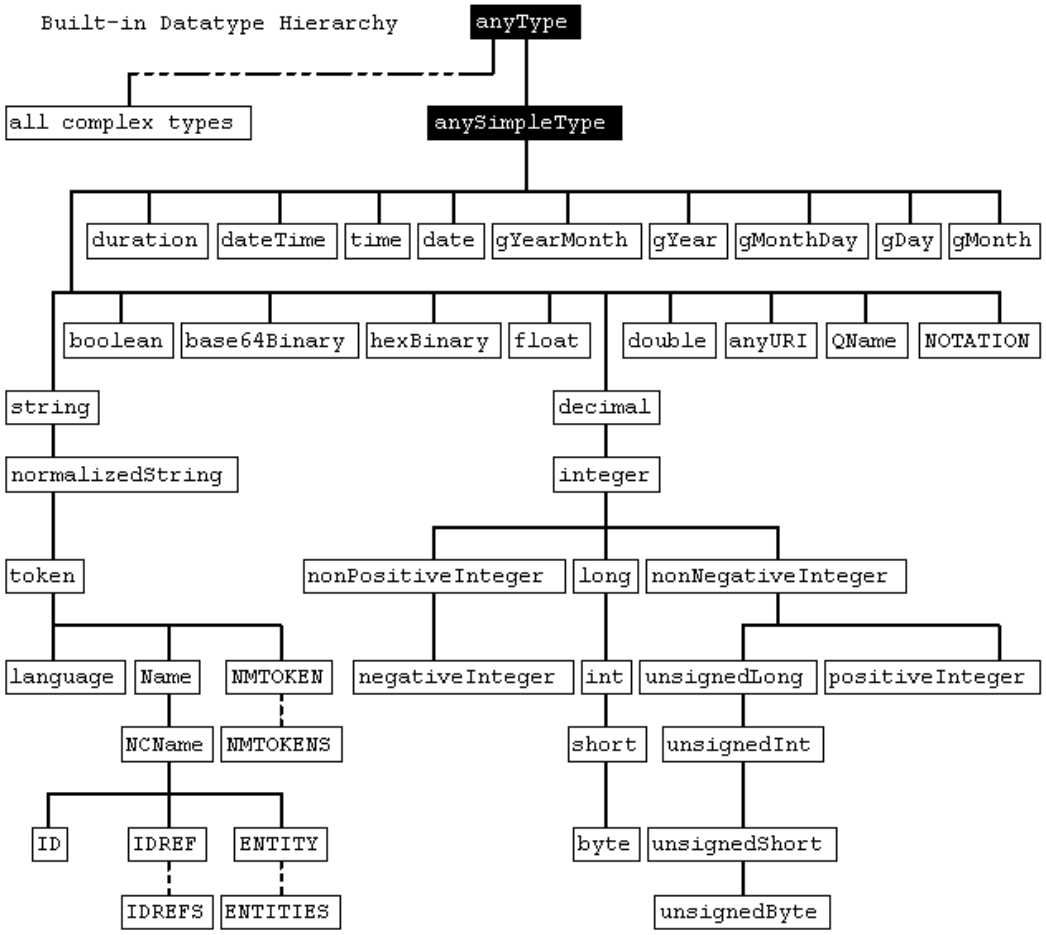
Výpis 7: XSD definice elementu

Jak je vidět, pro náš malý XML dokument je XSD větší než dokument samotný. XSD se snaží být hodně komplexní a bohužel díky tomu je také velice popisný.

Ve výpise (7) si může všimnout definice elementu pomocí `xs:element`. Příklad nám ukazuje, že element `firma` je složeným typem obsahujícím další elementy. `xs:sequence` definuje přesné pořadí elementů v něm uvedených (`nazev`, `adresa`, `telefon`). Typ `xs:string` znamená, že element obsahuje jako hodnotu řetězec. Tento typ patří mezi základně definované typy. Těchto typů, jak můžeme vidět na obrázku (1), je značné množství. Definují řetězcové hodnoty, číselné, logické, časové a další. V příkladech použijeme pouze několik nejběžnějších, o zbytku a jejich použití se můžeme dočíst v referenční příručce [5].

Deklarace atributů, taktéž využívá těchto typů a přidává několik možností restrikcí. Ty nám pomohou přesně specifikovat daný rozsah hodnot jednotlivých atributů. Samozřejmostí je možnost vzájemné kombinace několika restrikcí. Možnosti restrikcí jsou:

- omezení počtu znaků
- omezení pomocí regulárního výrazu
- omezení výčtem hodnot



Obrázek 1: Přehled základních XSD typů

- definice číselného rozsahu
- definice počtu desetinných míst
- definice počtu míst celočíselné hodnoty

Ve výpise (8) můžeme vidět jednoduchou definici několika atributů a jejich celkové provázání v rámci dokumentu. Všimněme si různých typů restrikcí, jako například `totalDigits` (rozsah bude 0 - 999), `lenght` (řetězec bude mít délku maximálně 25 znaků) a `enumeration` (řetězec bude nabývat pouze daných výčtových hodnot). V případě složeného typu pomocí `use` definujeme povinnost atributu a u definice elementu `zamestnanec` definujeme pomocí `minOccurs` a `maxOccurs`, že výskyt tohoto elementu musí být minimálně jednou, ale může se opakovat nesčetněkrát.

```
<!-- Povinná definice. -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Definice atributů. -->
<xs:simpleType name="idType">
  < xs:restriction  base="xs:nonNegativeInteger">
    < xs:totalDigits  value="3" />
  </ xs:restriction >
</xs:simpleType>

<xs:simpleType name="jmenoType">
  < xs:restriction  base="xs:string">
    <xs:lenght value="25" />
  </ xs:restriction >
</xs:simpleType>

<xs:simpleType name="oddeleniType">
  < xs:restriction  base="xs:string">
    <xs:enumeration value="IT" />
    <xs:enumeration value="UCETNI" />
  </ xs:restriction >
</xs:simpleType>

<!-- Definice složeného typu zamestnanec. -->
<xs:complexType name="zamestnanecType">
  <xs:attribute  name="id" type="idType" use="required" />
  <xs:attribute  name="jmeno" type="jmenoType" use="required" />
  <xs:attribute  name="oddeleni" type="oddeleniType" use="required" />
</xs:complexType>

<!-- Definice firmy. -->
<xs:element name="firma">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="zamestnanec" type="zamestnanecType" minOccurs="1"
                  maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>

<!-- Příklad validního XML dokumentu. -->
<firma>
    <zamestnanec id="1" jmeno="Petr␣Rybka" oddeleni="IT" />
</firma>
```

Výpis 8: XSD komplexní příklad

Opět lze vidět, že i na naprosto jednoduchý dokument s pár elementy a atributy, je XSD soubor poměrně rozsáhlý. Ukázali jsme si zde pouze jednoduché řešení, shrnutí celé problematiky je příliš rozsáhlé. Více v literatuře [4, 5] a na internetových stránkách W3C.

3.4 Soubory vlastního typu

Samozřejmě se můžeme rozhodnout nevybrat si žádný z dříve zmíněných typů a vytvoříme si vlastní.Tento způsob má své výhody a zároveň nevýhody. Mezi výhody patří, že si vytvoříme typ, který přesně potřebujeme. Můžeme si k němu také vytvořit šablonu pro jeho definici.

Bohužel nevýhod je celá řada. Musíme si celý typ naspecifikovat a musíme vytvořit jeho obsluhu pomocí některého z programovacích jazyků kompletně od začátku. Samozřejmě, pokud budeme chtít využít nějakého validátoru, budeme si jej muset udělat. Musíme také každého, kdo s ním přijde do styku, s tímto typem naučit pracovat. Celý tento proces je příliš časově náročný a vybrat si z již existujících typů, je mnohem jednodušší a pohodlnější volba. Jednak jsou všeobecně známé a také velice rozsáhle podporované. Což je zvláště při programové implementaci nespornou výhodou.

I přes to vše jsou stále situace, kdy se tvorba vlastního typu vyplatí a měli bychom alespoň zvážit, zda tento způsob nevyužít.

3.5 Shrnutí

Ukázali jsme si souborové typy INI, CSV a XML. Zmínili jsme také možnost tvorby vlastního datového typu.

Vzhledem k povaze zpracovávaných dat byl pro zpracování zvolen typ XML. Vybrán byl pro svou přehlednou strukturu a rozšířenou podporu v programovacích jazycích. Množství dat by při tom nemělo být natolik veliké, aby zpracování XML souboru činilo potíže. Pro jeho definici jsme použili DTD. I když DTD nyní prakticky uvolnilo místo XSD, stále se hodí pro definici jednoduchých dokumentů, jakými protokoly dat z výrobní linky bezesporu jsou. Díky DTD, můžeme dokumenty validovat a ušetřit si tak spoustu práce vytvářením kontroly pro jiné typy (i když jiné typy by byly úspornější z hlediska použitého diskového prostoru).

Podkapitola 3.3 byla rozsáhlá právě z důvodu výběru tohoto typu a k lepšímu pochopení spolupráce XML se schémovými jazyky.

4 Způsoby přenosu souborů

V předchozí kapitole jsme se zabývali výběrem vhodného typu datového souboru. Data uložená v tomto souboru musíme vhodným způsobem odeslat z počítače, na kterém běží EI, do EI serveru. Tímto se zabývá tato kapitola.

Obecně máme dvě varianty způsobu přenosu. Metodu PUSH a metodu PULL. Každá má své výhody a každá se hodí na něco jiného. Pokusíme se zde rozebrat jejich základní přednosti. Podíváme se na jejich výhody, nevýhody, potřeby instalace a administrace. Tento rozbor se vztahuje přímo k zamýšlené aplikaci.

4.1 PUSH přístup

Jde asi o nejčastěji využívaný přístup. Server jednotlivě přistupuje ke klientům (výrobním PC) a bere si potřebná data. Ty jsou u klienta někde uloženy na HDD. Jelikož server podobným přístupem nemůže brát ze všech klientů najednou, musí je postupně vybrat a data si vzít. Tato metoda má využití, pokud nechceme mít data k dispozici zrovna okamžitě.

Výhody:

- Žádné fronty - server by si sám bral data dle vlastní potřeby.
- Na klientském počítači by nemusela běžet žádná složitá aplikace a tak by byl minimálně zatěžován. Pokud ano, pak jen jednou za určitý čas, což by šetřilo jeho zatížení.
- V době mezi zpracováním dat, by aplikace na serveru vyvíjela prakticky minimální aktivitu na výrobních PC žádnou.

Nevýhody:

- Data nejsou okamžitě uživateli k dispozici.
- Samotný proces příjmu dat od klientů a jejich zpracování může být zdlouhavé při větším objemu dat, či velkém množství klientů.
- V čase mezi zpracováním dat, by aplikace na serveru vyvíjela prakticky minimální aktivitu na výrobních PC žádnou.
- V době zpracování může aplikace zabrat hodně z výkonu serveru. Pokud je server používán více aplikacemi, mohl by nastat problém.
- Pokud by se z nějaké příčiny na delší dobu přerušilo spojení serveru a klienta, klient přesto dál střádal data, mohl by nastat problém při následném zpracování. Zásoba dat k přenesení by mohla být tak velká, že by se za celou periodu nemuselo podařit přenést všechna data ze všech klientů. Situace by se dala řešit tak, že by server

přerušil download dat z aktuálního PC a začal zase od prvního PC. Předpokladem je, že by se jednalo o opravdu extrémní případ a perioda sběru dat by byla opravdu krátká.

Instalace:

- Na serveru bude potřeba aplikace, která bude řídit připojení ke klientskému PC, bude přenášet data a daným způsobem je evidovat (at' už v souboru nebo v DB).
- Na klientském PC bude muset být nainstalována jednoduchá aplikace pro komunikaci se serverem, ale v případě jednoduché síťové aplikace by mohla být porušena multiplatformnost. Jednou z dalších variant pro komunikaci je FTP.

Jelikož by se data vybírala v určitých časových úsecích, bylo by vhodné mít na klientském PC určitou repository, kde by se data skladovala v souborech. Pro jednodušší zpracování by bylo nejvhodnější data rozdělit na více malých souborů. Pokud by bylo třeba mít pro jistotu data zálohovány na klientském PC, byly by vhodné repository dvě. Z jedné by se data četla, zpracovávala a jakmile by byla v pořádku uložena na serveru, soubor by se přesunul do druhé repository (na místo zálohy). Šlo by o frontu, která by se zpracovávala, dokud by první repositora nebyla úplně prázdná. Server by se po té odpojil a pokračoval ve stejné proceduře s dalším klientským PC.

Jak často data vybírat? To by záleželo na počtu klientských stanic, průměrné době trvání výrobních procedur plus nějaká časová rezerva na zpracování.

Server by určitě měl kontrolovat zda data, jež dostal, jsou korektní. At' už fyzicky, tak syntakticky.

V případě vadných dat, by se měl server pokusit o jejich znovupřenesení. A pokud problémy přetrvají, na chybu se musí upozornit a zanést ji do logu. Celé monitorování by bylo na straně serveru.

Tento způsob by měl být spolehlivý, protože samotné spojení serveru a klienta nebude příliš dlouhé, to zmenšuje pravděpodobnost jeho selhání. Aplikace by se měla snažit o znovupřenesení dat v případě nějaké závady. V případě, že by se nedařilo vůbec navázat spojení, měl by se o to server pokusit vícekrát. Pokud ani pak neuspěje, pokusí se o přenos v další periodě.

4.2 PULL přístup

Tento způsob přístupu funguje na naprosto odlišném principu než způsob přístupu PUSH. Klienstské stanice samy nabízejí data serveru. Ten si je přebírá a dále je zpracuje. Takto se mohou data odesílat hned jakmile jsou k dispozici. Dalo by se říci v reálném čase (to je ale spíše optimistické tvrzení).

Výhody:

- Možnost téměř okamžitého zpracování dat.

- Časté odesílání malých úseků vede k rychlému zpracování.
- Možnost využít webové služby.

Nevýhody:

- Nutnost řešit situaci fronty, kdy více klientů chce nabízet data ve stejnou dobu (někdy je programově ošetřena sama).
- Častá komunikace mezi serverem a klientem.
- Nepřetržité využití určité části výkonu klienta pro aplikaci hlídající nová data a komunikující se serverem.
- Při využití FTP by byla komunikace složitější, pokud by došel soubor s vadnými daty, protože by se opět musela kontaktovat aplikace u klienta, aby daný soubor znova poslala. Komunikace by tedy byla hustší.

Instalace:

- Aplikace pro zpracování dat na serveru.
- Aplikace pro komunikaci se serverem - na klientském PC.
- Aplikace pro zajištění běhu webových služeb na serveru (v případě jejich použití).
- (V případě využití FTP, jeho instalace na server.)

Jakmile by klient dostal potřebná data, vytvořil by z nich soubor, upozornil by server a soubor mu předal. Server by soubor zpracoval, v případě chybného zpracování, by si vyžádal soubor znovu a pokusil se jej znova přenést. Pokud by vše bylo v pořádku, soubor by se u klienta zálohoval. A aktivní komunikace by se ukončila.

V případě využití FTP by klientské PC přenesly nová data ihned na server a vyčkávaly by. Server by data zpracoval a dle úspěšnosti by si je nechal přeposlat, nebo by odeslal zprávu, že je vše v pořádku. Pak by si klient data zálohoval a byl by pasivní, zase do doby než bude mít nová data. Pokud by naráz došlo serveru velké množství dat, zpracovával by je postupně a to by mohlo mít za následek čekací dobu, po kterou by k němu bylo připojeno hned několik klientů a ti by vyčkávali. Samozřejmě tato doba by se úměrně zvyšovala s množstvím dat.

Místo použití FTP by se vhodně dalo využít WS (webových služeb). Server by jednoduše naslouchal a jakmile by se klient připojil k dané webové službě, vytvořil by její novou instanci, data přijal a zpracoval. Výsledek zpracování by se obratem vrátil zpět ke klientovi a ten by věděl, zda má data znovu zaslat či ne. Velikou výhodou WS je, že se dají nasadit i na místech, kde je běžná síťová komunikace velice obtížná. Zpracování by zajišťoval server a na klientské stanici by byl pouze jednoduchý program pro komunikaci s danou webovou službou.

Opět by byl důležitý nějaký logovací soubor a kontrola korektnosti došlých dat. V případě několikanásobného selhání přenosu dat by se měla celá událost zalogovat a měla by na sebe vhodným způsobem upozornit.

Data by se dostávala na server průběžně a rychle. V případě nějaké poruchy aplikace na klientském PC, není server schopen data žádným způsobem získat, protože iniciátorem zpoždění je klient, server by neměl ani zájem data získávat.

4.3 Výběr přístupu

Jako jeden z požadavků máme platformní nezávislost. Dále si musíme uvědomit, že výrobní PC nejsou příliš výkonné a zároveň je jejich softwarová výbava pod přísným dohledem. Přesto by se daly využít obě z výše popsaných metod.

Metoda PUSH by se dala využít krásně s použitím FTP. FTP server je jednoduchou aplikací, která příliš nezatěžuje počítač a dala by se využít i na výrobních PC. Velikou nevýhodou je prodleva mezi jednotlivými zpracováními dat. Dodané informace by se na server mohly dostávat se značným zpožděním.

Metoda PULL nabízí rychlé zpracování informací s minimální aktualizační prodlevou. Zde by využití FTP bylo poněkud komplikovanější. Naštěstí tento problém řeší využití webových služeb, které mohou zajistit rychlou, bezproblémovou a spolehlivou komunikaci.

Zvolili jsme metodu PULL s využitím webových služeb. Vybrána byla pro rychlost a zároveň pro bezproblémovost přenosu. S použitím protokolu HTTP by se měly dát dobře obejít všechny možné překážky v rámci komunikační infrastruktury firemní sítě.

5 Uložení dat

Datové soubory, které dostane server od klientů je nutné zpracovat a data nějakým způsobem vhodně uložit. Přitom k nim stále potřebujeme mít snadný přístup, abychom v nich mohli vyhledávat. Nabízejí se nám dva způsoby, jak toto uložení provést. Agendové zpracování dat, nebo Databázové zpracování dat. My jsme použili Databázové zpracování dat. Agendové zpracování zde uvádíme pouze pro přehled.

5.1 Agendové zpracování dat

Jde o uložení dat pomocí námi napsaného programu, který bude provádět pouze jednoduché operace, jež mu implementujeme. Vše si musíme napsat sami, navrhnout uložení dat, jejich formu a jak se k nim bude přistupovat. Vhodné spíše pro malé množství dat u kterých se nepředpokládají inovace struktury.

Výhody:

- Vše bude navrženo jak chceme.
- Je pravděpodobné, že se ušetří prostor.

Nevýhody:

- Velice obtížná manipulace s daty.
- Při nevhodném navržení, zdlouhavé vyhledávání záznamu.
- V případě změny charakteru dat, se musí udělat změny v celém programu (žádná flexibilita).
- Tento způsob není vhodný pro přístup více uživatelů.

5.2 Databázové zpracování dat

Dnes nejrozšířenější, není potřeba nic psát. O manipulaci s daty se stará daný SŘBD. Nabízí spoustu funkcí jak s daty pracovat. Relační model pracuje s daty jako s tabulkami a je dostatečně pružný.

Výhody:

- Snadná instalace a administrace.
- Skvěle zvládnutý přístup pro více uživatelů.
- Rychlé vyhledávání.
- Stabilita.

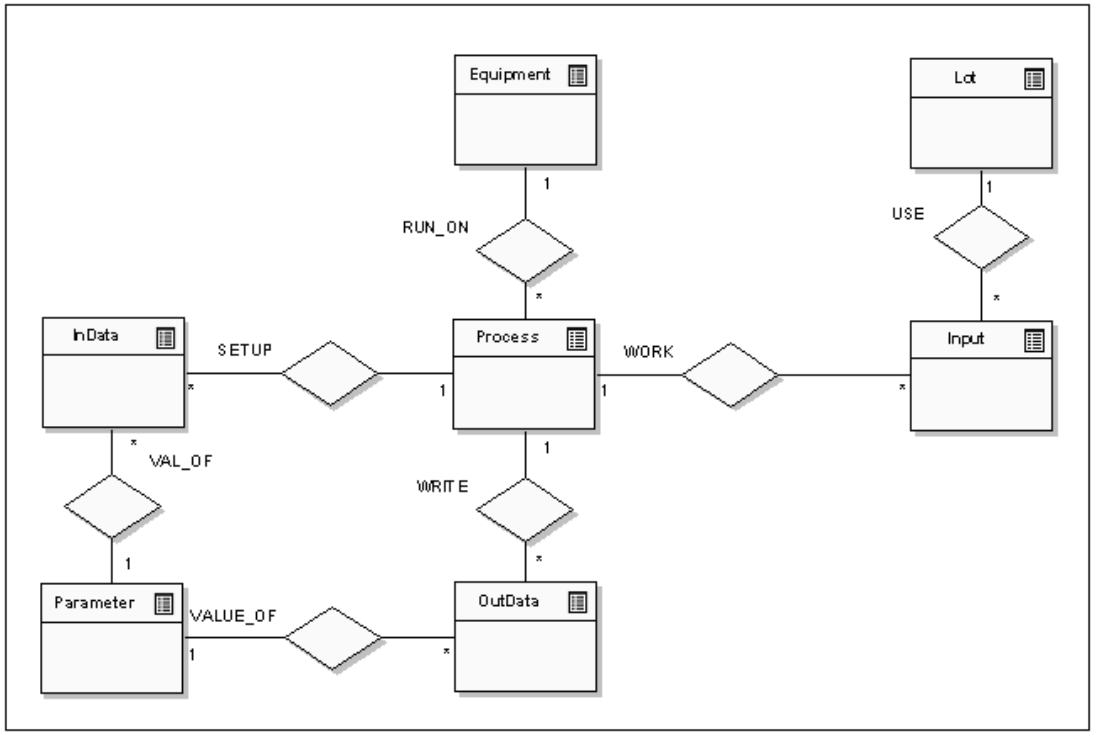
- Bezpečnost.
- Snadná správa velikého množství dat.

Nevýhody:

- Někdy HW náročnost.

5.2.1 Struktura databáze

Předpokládá se, že z výrobních PC se bude dostávat veliké množství naměřených dat. Agendové zpracování tedy nepřipadá v úvahu. Použijeme databázové zpracování na SŘBD Oracle.



Obrázek 2: ER diagram

Na obrázku (2) můžeme vidět ER diagram databáze. Jak je vidět databáze není příliš složitá. Jednotlivé vazby mezi tabulkami nalezneme v tabulce (5).

Samozřejmě samotný ER digram nám příliš neprozradí, čeho se která tabulka týká. Pro vysvětlení si jednotlivé vstupy popíšeme. Ještě před tím je třeba zmínit, že následující struktura je pouze simulativní a konečná podoba nasazená v praxi, může být ještě doladěna.

Název	Tabulky	Vazba
RUN_ON	Equipment, Process	1:N
SETUP	Process, InData	1:N
USE	Lot, Input	1:N
VAL_OF	Parameter, InData	1:N
VALUE_OF	Parameter, OutData	1:N
WORK	Process, Input	1:N
WRITE	Process, OutData	1:N

Tabulka 5: Vazby

První tabulka (*Equipment*), eviduje `EQIPMENT_ID`, `CHAMBER_ID` a k tomu umělý primární klíč `equipment_table_id`, který bude jednoduchý a nemusíme nikde mnohokrát evidovat předchozí dva atributy. Ty mohou být hodně velké.

V druhé tabulce (*Process*) evidujeme `equipment_table_id` (z tabulky *Equipment*, tak přiřadíme jednoznačně equipment a komoru procesu), `RUN_MODE` (mód daného běhu - run, iddle), `RUN_ID` (id daného běhu), `PP_ID` (id daného procesního programu), `PROCESS_START` (čas začátku procesu), `PROCESS_FINISH`(čas konce procesu). K tomu je třeba přidat atribut `process_table_id`, který bude primárním klíčem (důvod je jednoduchý, později ho využijeme jako foreign key k snadnému provázání dat a hlavně potřebujeme unikátně dělit záznamy).

Ve třetí tabulce (*Lot*) evidujeme informace o lotech. U každého záznamu (lotu), budeme evidovat `LOT_ID`, `BATCH_ID` (tak jednoduše zjistíme zda je lot v nějaké várce) a nakonec přidáme umělý primární klíč `lot_table_id`.

Čtvrtá tabulka (*Input*) slouží jako vazební tabulka k rozdělení vazby M:N mezi tabulkami *Process* a *Lot*. Proto jsou v ní evidovány 2 cizí klíče z obou jmenovaných tabulek, které dohromady tvoří složený primární klíč. Jde o atributy `process_table_id` a `lot_table_id`.

Pátá tabulka (*Parameter*) eviduje atributy `PARAMETER_NAME` (jméno parametru), `PARAMETER_TYPE` (typ parametru) a k nim umělý primární klíč `parameter_table_id`. Jde o číselník parametrů.

Šestá tabulka (*InData*) obsahuje informace o vstupních parametrech procesu. Eviduje se hodnota daného parametru `PARAMETER_IN_VALUE` a pomocí cizího klíče `parameter_table_id` přiřadíme tuto hodnotu k danému parametru z číselníku. Abychom to vše mohli přiřadit i k procesu, použijeme cizí klíč `process_table_id`. Složený primární klíč utvoříme z atributů `parameter_table_id` a `process_table_id`.

Sedmá tabulka (*OutData*), eviduje hodnoty parametrů v daných časech. Tedy `PARAMETER_OUT_VALUE` a přiřadíme pomocí cizího klíče z tabulky parametrů - `parameter_table_id` k danému parametru. Dále potřebujeme cizí klíč `process_table_id` jako přiřazení k procesu a ještě nesmíme zapomenout na atribut `TIME_STAMP`, který udává, kdy byla hodnota sejmuta. Na závěr doplníme umělý primární klíč `outdata_table_id`.

6 Webové služby

Komunikace, at'už v rámci podnikových sítí nebo širších oblastech není vždy jednoduchá. Navázat jednoduché socketové spojení může být značným problémem. Můžou nám bránit různé překážky už při vytváření samotného spojení - proxy servery, firewally, atd. Internet jako celosvětová informační síť je dostupná skoro všude. Jedná se o čistě platformě nezávislou záležitost. Nezáleží kde a na jakém operačním systému k němu přistupujeme. Protokol HTTP vždy poskytne dané informace. Díky rozšíření internetu se dá také předpokládat, že služby tohoto protokolu nebudou ve většině sítí blokovány. Toho se snaží využít tzv. Webové služby (Web services). Webové služby díky protokolu HTTP přenášejí zprávy v jazyce XML. S jejich pomocí můžeme aplikovat RPC (Remote Procedure Call), česky vzdálené volání procedur.

Můžeme tedy zasílat různá data ke zpracování vzdálenému serveru a dostaneme zpět nějaký výsledek. Standardem v dnešní době jsou webové služby s využitím protokolu SOAP. Tento způsob je popsán ve specifikaci W3C [6]. Roy Thomas Fielding se ve své disertační práci [7] zabýval využitím webových služeb na architektuře REST (Representational state transfer). Využití této architektury celé využití webových služeb značně zjednodušuje a oprošťuje definici webové služby o popisný jazyk WSDL, který si popíšeme dále a využívá více možností protokolu HTTP.

My se zde budeme zabývat klasickými webovými službami podle specifikace W3C. Postupně si popíšeme, co vše je potřebné k provozování webových služeb.

6.1 SOAP

Simple Object Access Protocol je komunikační protokol. S jeho pomocí komunikují aplikace prostřednictvím internetu. Jedná se o formát pro posílání zpráv. Je založen na jazyce XML. Mezi jeho přednosti patří platformní a jazyková nezávislost schopnost projít přes firewally. Spadá také do rodiny doporučení W3C [8].

Zkrátka umožňuje nám vzdáleně volat různé funkce. Pomocí protokolu HTTP se přenese „obálka“ s XML zprávou odkazující na danou volanou funkci (metodu). Po provedení dané akce se posílá naprosto stejným způsobem odpověď.

Jak komunikace vypadá si ukážeme na ukázkách převzatých z tutoriálu W3C[8].

```
POST /InStock HTTP/1.1
Host: www.example.org
Content−Type: application/soap+xml; charset=utf−8
Content−Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap−envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap−encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
```

[8] W3C SOAP tutorial - http://www.w3schools.com/soap/default.asp

```
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

Výpis 9: Příklad SOAP dotazu

Ve výpise (9) vidíme příklad dotazu na metodu `GetStockPrice` s parametrem `StockName` a jeho hodnotu `IBM`. Metoda se nachází na adrese *http://www.example.org/stock*

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>
</soap:Envelope>
```

Výpis 10: Příklad SOAP odpovědi

Výpis (10) uvádí příklad odpovědi na dotaz ve výpise (9). Tedy metoda `GetStockPrice` nám vrací na předchozí dotaz hodnotu parametru `Price=34.5`

S těmito dotazy a odpověďmi se většinou přímo nesetkáme, avšak kdybychom zachytili komunikaci pomocí nějakého síťového analyzéru (Ethereal), dostali by jsme právě podobný dotaz a odpověď.

6.2 WSDL

Web Services Description Language (WSDL) je popisný jazyk využívaný ve webových službách. Je to právě on, který popisuje jak a co se má volat. Co může popisovat vidíme v tabulce (6). Operačních typů také může být několik. Defaultně pracujeme s typem `request-response`, kdy operace zprávu přijme a nazpět zašle odpověď. Jsou i další typy. Například se zpráva přijme, ale na odpověď se již nečeká. Nebo operace pošle zprávu a čeká na odpověď.

WSDL nám tedy popisuje, jak ke vzdáleným operacím přistupovat, jaké typy atributů používají, jaké vrací chybová hlášení a výsledky.

WSDL soubor je XML dokument poskytovaný všem klientům dané webové služby. Můžeme jej vytvořit ručně, ale webové služby mají podporu různých IDE (NetBeans, Eclipse, ...) a tyto nám mohou WSDL k dané službě automaticky vygenerovat. Šetří tím spoustu času stráveného samotným psaním a opravami chyb.

Element	Popis
<types>	Datové typy použité webovou službou.
<message>	Zpráva použitá webovou službou.
<portType>	Operace poskytované webovou službou.
<binding>	Komunikační protokol použitý webovou službou.

Tabulka 6: Elementy ve WSDL pro popis webové služby

6.3 Implementace v jazyce Java

Pro provozování webové služby potřebujeme nejdříve nainstalovat a nakonfigurovat kontejner pro aplikaci umístěnou na serveru. Můžeme například použít GlassFish nebo Apache Tomcat.

Na serveru vytvoříme webovou aplikaci. Výpis (11) ukazuje příklad definice webové služby `Kalkulacka`, která má pouze jedinou operaci `secti`. Ta sečte dvě vstupní celá čísla a vrátí výsledek. Všimněme si, že webovou službu tvoříme jako třídu.

```
package somePackage;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

@WebService()
public class Kalkulacka {
    /**
     * Web service operation
     */
    @WebMethod(operationName = "secti")
    public int secti(@WebParam(name = "a")
    int a, @WebParam(name = "b")
    int b) {
        return a+b;
    }
}
```

Výpis 11: Tvorba jednoduché webové služby

Volání operace `secti` u klientské aplikace provedeme metodou ve výpise (12). Po zavolání této metody dostaneme na standardním výstupu výsledek `7`. Jak vidíme, tvorba jednoduché webové služby není příliš složitou záležitostí. Mnohá IDE nám zkrátí celou tvorbu na vyplnění několika formulářů a naprogramování logiky operace.

```
public void vypocti () {
        try { // Call Web Service Operation
            somepackage.KalkulackaService service = new somepackage.KalkulackaService();
            somepackage.Kalkulacka port = service.getKalkulackaPort();
            int a = 3;
            int b = 4;
            int result = port.secti(a, b);
```

```
            System.out.println("Result = "+result);
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
```

Výpis 12: Klient jednoduché webové služby

Pro celý příklad nám IDE automaticky vygenerovalo WSDL dokument ve výpise (13).

```
<definitions targetNamespace="http://somePackage/" name="KalkulackaService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://somePackage/" schemaLocation="http://localhost:8084/
                Server/Kalkulacka?xsd=1"></xsd:import>
        </xsd:schema>
    </types>
    <message name="secti">
        <part name="parameters" element="tns:secti"></part>
    </message>
    <message name="sectiResponse">
        <part name="parameters" element="tns:sectiResponse"></part>
    </message>
    <portType name="Kalkulacka">
        <operation name="secti">
            <input message="tns:secti"></input>
            <output message="tns:sectiResponse"></output>
        </operation>
    </portType>
    <binding name="KalkulackaPortBinding" type="tns:Kalkulacka">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"></
            soap:binding>
        <operation name="secti">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
        </operation>
    </binding>
    <service name="KalkulackaService">
        <port name="KalkulackaPort" binding="tns:KalkulackaPortBinding">
            <soap:address location="http://localhost:8084/Server/Kalkulacka"></soap:address>
        </port>
    </service>
</definitions>
```

Výpis 13: Vygenerované WSDL

7 Zpracování XML

Způsobů práce s XML v Javě máme několik. Stromovou reprezentaci dokumentu, typický představitelem je DOM (Dokument Object Model) nebo událostmi řízené zpracování SAX (Simple API for XML).

DOM pracuje, tak že načte dokument do paměti jako stromovou strukturu. Práce s dokumentem je pak pohodlná, používáme objekty. Můžeme se pohybovat mezi elementy dokumentu a jejich atributy dle libosti. Bohužel velikou nevýhodou je zpracování rozsáhlých XML dokumentů. Uložený dokument v paměti zabírá spoustu místa a může se stát, že pro dokument o velikosti řádově desítek až stovek megabytů, nám nebude stačit operační paměť.

SAX neukládá dokument do paměti. Postupně jej čte a reaguje na události vyvolané danými elementy a atributy. Je vhodný pro práci s velikými dokumenty, protože oproti DOMu nezatěžuje tolik operační paměť počítače. Velikou nevýhodou je, že při čtení dokumentu se už nemůžeme vracet na dříve přečtené elementy a atributy.

Pro zpracování daného úkolu jsme zvolili SAX. Předem nevíme jak rozsáhlé budou zpracovávané dokumenty. Známe pouze jejich strukturu a záleží nám na rychlosti zpracování. SAX je tedy vhodným řešením pro tento úkol.

7.1 SAX

Samozřejmě, že parser nemusí jen dokument rozparsovat na jednotlivé části. Můžeme využít několik dalších nastavení. Nejdůležitějším z nich je validace oproti DTD, XSD. Validaci dokumentu využíváme poměrně často. Ve výpise (14) vidíme, jak se provede parsování dokumentu s validací.

```
public boolean validateXML(){
    SAXParserFactory spf;
    SAXParser sp = null;
    FileReader file = null;
        try {
            // nastavení cesty XML dokumentu
            file = new FileReader(new File("soubor.xml"));
            InputSource input = new InputSource(file);
            spf = SAXParserFactory.newInstance();

            // validace povolena
            spf.setValidating(true);

                //nastavení parseru
            sp = spf.newSAXParser();
            parser = sp.getXMLReader();

            // přetížení původní cesty k DTD v daném dokumentu novou cestou k DTD
            parser.setEntityResolver(new EntityResolver() {

                public InputSource resolveEntity(String publicId, String systemId) throws
                    SAXException, IOException {
                return new InputSource("soubor.dtd");
```

```
                }
            });
            // spuštění parseru
            parser.parse(input);
            return true;
        }
        catch (Exception e)
        {
          ...
        }
    }
```

Výpis 14: SAX s nastavením validace

Celkově se nejedná o nikterak složitou záležitost. Všimněme si přetížení zdrojového DTD souboru. To se nám může hodit v případě, kdy je dokument vytvořen někde vzdáleně na jiném PC a na našem PC je cesta k danému DTD definovanému přímo v dokumentu jiná. Tímto způsobem parteru vnutíme, aby použil naši novou cestu.

Zpracování XML dokumentu provádíme podobným způsobem. Musíme dokumentu přiřadit vlastní `ContentHandler` obstarávající události vyvolané proudový zpracováním dokumentu. Tedy při přečtení nějakého elementu se vyvolá událost s ním svázaná. Například přidání nějakého záznamu do protokolu apod. Přiřazení tohoto handleru se provádí takto:

```
parser.setContentHandler(new MujHandler());
```

Toto nastavení se samozřejmě přidává ještě před spuštěním parseru. Musíme mít tedy vytvořenu třídu `MujHandler` rozšiřující `DefaultHandler`. Jaké metody například pak k reakci na jednotlivé události používáme, můžeme vidět ve výpise (15). Pozor, nejedná se o příklad popisující všechny použitelné metody.

I když se zdá obsluha událostí v parseru snadná, měli bychom pamatovat, že při změně struktury parsovaného XML dokumentu, bychom neměli zapomenout poupravit i obsluhu tohoto dokumentu. Nehodí se tedy pro příliš časté změny.

```
public class MujHandler extends DefaultHandler {
    private StringBuffer currentContent;

      // Constructor
    public MujHandler(){
        currentContent = new StringBuffer(10000);
    }

    @Override
    public void characters(char[] ch, int start, int length) throws SAXException {
        currentContent.append(ch, start, length);
    }

    // metoda obsluhující koncové tagy elementů
    @Override
    public void endElement(String uri, String localName, String qName) throws SAXException {
        if (qName.equals("zelenina") == true){
```

```
                ...
            }
        }

        // metoda volaná na konci dokumentu, je vhodné ji použít například k odeslání získaných dat
            dále
        @Override
        public void endDocument() {
            ...
        }


        // metoda pro obsluhu elementů
        @Override
        public void startElement(String uri, String localName, String qName, Attributes attributes)
            throws SAXException {
            if (qName.equals("zelenina")) {
                // vypíše hodnotu atributu "jmeno" v elementu "zelenina"
                System.out.println(attributes.getValue("jmeno"));
            }

            ...
    }
```

Výpis 15: SAX příklad obsluhy událostí

8 Technické řešení

Samotné řešení problému je složeno z několika komponent. Spolupráci jednotlivých prvků vidíme na obrázku (3).

Výrobní počítače zpracovávají data z výroby do XML dokumentu. Běží na nich klientská aplikace pro spojení s webovou službou na serveru. Jakmile je XML dokument celý, klientská aplikace jej pomocí webové služby odešle na server ke zpracování (*krok 1*).

Na serveru v serverovém aplikačním kontejneru máme umístěnu aplikaci s webovou službou pro příjem XML dokumentů z výrobních PC. Ta dokument přijme a v *kroku 2* jej uloží na pevný disk. Dokument se pak zvaliduje a pokud projde, vygeneruje se odkaz na jeho zpracování další aplikací na serveru. Aplikace následně zašle v *kroku 3* odpověď s výsledkem. Záleží už na nastavení klientské aplikace, zda se pokusí zaslat dokument znovu, v případě chyby.

Dokumenty, které prošly validací, jsou na serveru dále předány ke zpracování aplikací k uložení do databáze. Provede se parsování dokumentu pomocí SAX parseru (*krok 4*) a vloží se patřičné údaje z výroby do databáze (*krok 5*) .

Chceme-li výrobní data následně zobrazit, spustíme internetový prohlížeč (Internet Explorer, Firefox, Opera, Netscape, atd.) a zadáme adresu Viewer aplikace běžící na serveru. V *kroku 6* předáme této aplikaci data, která chceme zobrazit. Viewer následně (*krok 7*) vybere daná data a zobrazí je procesnímu inženýrovi (*krok 8*).

Několik aplikací nám dohromady dá celý systém. Pro zajištění platformí nezávislosti jsou psány v Javě. Nezávislost na struktuře firemní sítě nám zajišťují webové služby. Předchozí popis nám ukázal, jak vše funguje jako celek. Nyní se podíváme trochu podrobněji na samotnou stranu serveru a stranu klienta (výrobního PC).

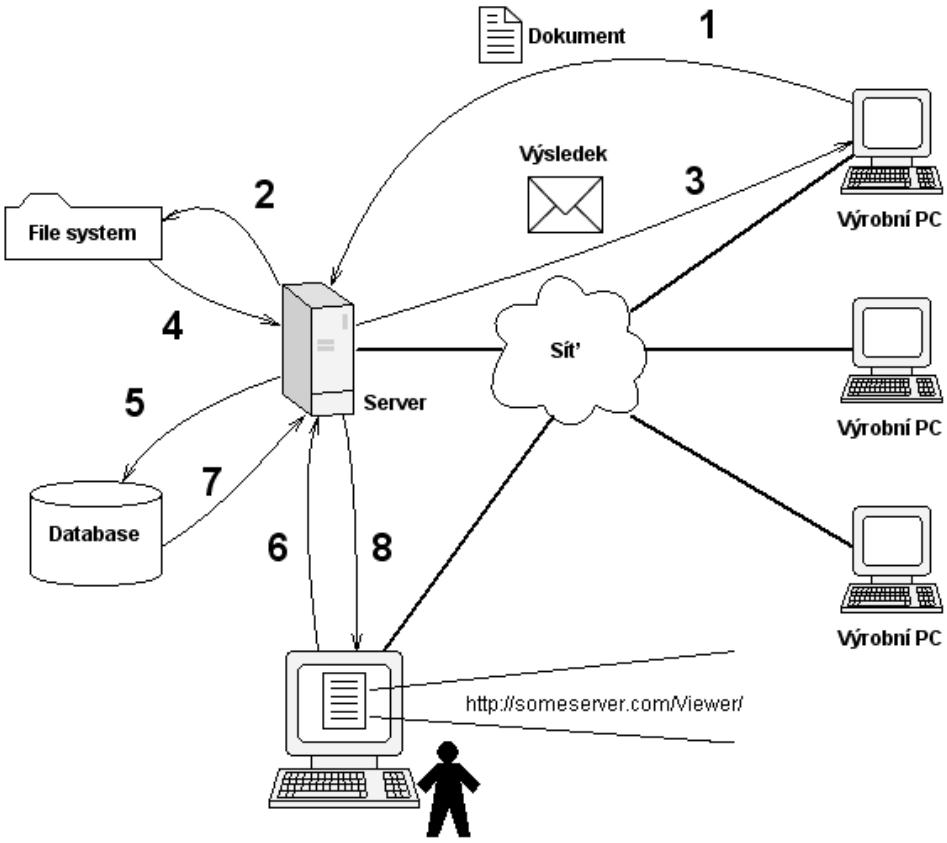
8.1 Server

Pro umístění aplikací na serveru potřebujeme nějaký aplikační kontejner. Můžeme využít GlassFish, JBoss, Apache Tomcat a jiné. Pro naše řešení jsme vybrali Apache Tomcat verze 6.0.18. Není náročný a nezatěžuje příliš hardware serveru. Cesta a port aplikačního kontejneru jsou důležité pro připojení se k našim serverovým aplikacím. Aplikace může pak mít cestu http://someaddress:8084/SomeApplication/

Na serveru nám poběží tři aplikace. Dvě komunikují s klienty a třetí aplikace běžící lokálně na serveru se stará o uložení dat do databáze.

Repository

Tato aplikace je klasickou serverovou variantou. Její součástí je webová služba, která naslouchá a vyčkává na volání klientské aplikace. Jakmile je vyzvána, přebere XML dokument jako pole bytů a na své straně jej opět poskládá v plnohodnotný XML dokument (dokumenty jsou členěny podle data a názvu výrobního PC). Provede se validace dokumentu. Když dokument vyhoví, uloží se odkaz na něj do souboru pro aplikaci zajišťující jeho uložení do databáze. Pokud dokument nevyhoví, přesune se do speciální



Obrázek 3: Technické řešení

složky určené pro smazání a celá událost bude zanesena do losovacího souboru. Na závěr klientská aplikace dostane výsledek generovaný webovou službou (true/false).

DB

Jedná se o aplikaci, která běží neustále na pozadí a kontroluje, zda v její zdrojové složce nejsou nové soubory odkazující na nové XML dokumenty. Kontrola se provádí periodicky v intervalu nastaveném pomocí properties souboru. V případě nových souborů, se tyto přečtou, načtou se z nich cesty k daným XML dokumentům a ty se pomocí SAX parseru zpracují. Zpracovaná data se následně vloží do databáze. Přístupy k složkám a konfiguraci připojení k SŘBD je nutné nastavit v properties souboru.

Viewer

Webová aplikace zobrazující JSP stránku v prohlížeči. Na této stránce lze pomocí vyplnění a odeslání formulářů vyhledat dané parametry k určitému procesu a zobrazit

je. K jednoduchému grafickému zobrazení je využita komponenta JFree chart. Aplikace jednoduše vyhledá v databázi potřebné údaje dle zadaných parametrů a výsledek poskytne ve formě JSP stránky uživateli (procesnímu inženýrovi). I u této aplikace je povinností nakonfigurovat správně připojení k SŘBD pomocí properties souboru.

8.2 Klient

Na straně klienta (výrobní PC) je situace mnohem jednodušší. Běží zde pouze jediná aplikace, která zasílá XML dokumenty pomocí webové služby na serverovou aplikaci `Repository`. Pro správné připojení k serverové aplikaci je potřeba nastavit správnou cestu v properties souboru.

9 Závěr

Jelikož byl daný projekt zpracováván přímo pro firmu, tato práce se zabývá obecným přehledem použitých technologií a přístupů. Snaží se přiblížit použité technologie a pomocí příkladů ukázat jejich využití. Díky této práci jsem se mohl naučit mnoho o jazyce XML a schémových jazycích DTD, XSD. Zjistil jsem výhody a nevýhody každého z uvedených schémových jazyků a zvolil vhodný pro daný projekt. Potřeba zpracovat XML dokumenty, mě naučila využít SAX parser v jazyce Java. Měl jsem možnost porovnat PUSH a PULL přístup a vybrat nejvhodnější řešení pro daný problém. Vyzkoušel jsem si spolupráci mezi Javou a SŘBD Oracle. Dalším velice zajímavým tématem bylo použití webových služeb, které jsou v dnešní době velice hojně využívány. Nastudování a jejich využití považuji za veliký přínos tohoto projektu.

Rozhodně by se dalo v projektu udělat několik dalších rozšíření. Velikou výhodou by mohlo být rozšíření o možnost exportu dat do různých datových formátů (CSV, INI, PDF, atd.), kterého by mohl využít uživatel využívající jednoduchý Viewer.

Viewer by určitě mohl mít i několik dalších funkcí, které by z něj mohly učinit silnější nástroj splňující více požadavků procesních inženýrů. Až nasazení přímo do výroby ukáže, kterých dalších funkcí by bylo potřeba.

Petr Hanták

10 Reference

[1] Kosek, J.: *XML pro každého - podrobný průvodce*. Grada Publishing, Praha 2000. 164 s. ISBN 80-7169-860-1.

[2] Herout, P.: *Java a XML*. Kopp nakladatelství, České Budějovice 2007. 313 s. ISBN 978-80-7232-307-4.

[3] Bray, T.: *Introduction to the Annotated XML Specification*. 1998. http://www.xml.com/axml/testaxml.htm

[4] Sperberg-McQueen, C. M. – Thompson, H.: *W3C XML Schema*. 2000. http://www.w3.org/XML/Schema

[5] Fallside, C. D. – Walmsley, P.: *XML Schema Part 0: Primer (Second Edition)*. 2004. http://www.w3.org/TR/xmlschema-0/

[6] Booth, D. – Haas, H.: *Web Services Architecture*. 2004. http://www.w3.org/TR/ws-arch/

[7] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[8] Mitra, N. – Lafon, Y.: *SOAP Version 1.2 Part 0: Primer (Second Edition)*. 2007. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

[9] Booth, D. – Liu, C. K.: *Web Services Description Language (WSDL) Version 2.0 Part 0: Prime*. 2007. http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/

A CD-ROM disc

Na přiloženém disku se nachází tato práce v podobě elektronického textového dokumentu (formát PDF). Také jsou v komprimovaném ZIP archivu přiloženy veškeré soubory použitých obrázků spolu se zdrojovým souborem této práce pro LaTeX. V archivu je také obsažen soubor poslední verze makra Diploma, které vytvořil pan doc. Mgr. Jiří Dvorský, Ph.D.